THE EMBODIMENTS OF THE INVENTION IN WHICH AN EXCLUSIVE PROPERTY OR PRIVILEGE IS CLAIMED ARE DEFINED AS FOLLOWS:

1. A method for determining the correctness of a potential interprocedural dead store optimization for an optimizing compiler, the optimizing compiler generating an intermediate representation of code to be compiled comprising a call graph, the method comprising a top- down traversal of the call graph, and comprising, for each procedure definition reached in the call graph traversal, the following steps

    a. defining a live on exit set of variables for the reached procedure definition, and

    b. defining a live on exit set of variables for each procedure call point within the reached procedure definition,

    c. storing the live on exit set of variables for each procedure call point in an entry in a live on exit data structure, and

    d. using the live on exit set of variables for the reached procedure definition to determine the variables that are ineligible for interprocedural dead store elimination in the reached procedure definition.

2. The method of claim 1 in which the live on exit set of variables for the reached procedure definition is defined by taking the union of all stored entries in the live on exit data structure corresponding to call points for the reached procedure.

3. The method of claim 2 in which the step of defining the live on exit set for each procedure call point in the reached procedure definition further comprises the steps of

    a. defining a basic block live set for each block of computer code in a control flow graph for the reached procedure definition, the basic block live set comprising the variables used in the block of computer code and the variables used in any procedure called within the block of computer code, and

b. determining the live on exit set for each procedure call by taking the union of the basic block live sets for all successor blocks to the block in the control flow graph containing the procedure call point and by adjusting the union to include uses of variables in the code between the call point for the procedure and the end of the block containing the call point.

4. The method of claim 1 in which the step of defining the live on exit set for each procedure call point in the reached procedure definition further comprises the steps of

    a. defining a basic block live set for each block of computer code in a control flow graph for the reached procedure definition, the basic block live set comprising the variables used in the block of computer code and the variables used in any procedure called within the block of computer code, and

    b. determining the live on exit set for each procedure call by taking the union of the basic block live sets for all successor blocks to the block in the control flow graph containing the procedure call point and by adjusting the union to include uses of variables in the code between the call point for the procedure and the end of the block containing the call point.

5. The method of claim 2, further comprising the step, after defining the live on exit set of variables for the reached procedure definition, of removing all entries in the live on exit data structure corresponding to call points for the reached procedure.

6. The method of claim 3, in which the variables used in a procedure called within a block of computer code are determined by accessing the mod/use set for the procedure associated with the procedure definition node in the call graph.

7. The method of claim 1 in which the step of using the live on exit set of variables for the reached procedure definition to determine the variables that are ineligible for interprocedural dead store elimination in the reached procedure definition comprises the step of generating pseudo uses of the members of the live on exit set of variables for the reached procedure definition in the data flow graph for the reached procedure definition.

8. The method of claim 1 in which the live on exit set data structure comprises bit vector entries and is indexed by call graph edges.

9. The method of claim 2 further comprising the step of using the live on exit set of variables for the procedure definition to determine whether the procedure definition may be cloned by the optimizing compiler.

10. A method for determining the correctness of a potential interprocedural dead store optimization for an optimizing compiler, the optimizing compiler generating an intermediate representation of code to be compiled comprising a call graph, the method comprising a top- down traversal of the call graph, and comprising, for each procedure definition reached in the call graph traversal, the following steps

    a. defining a live on exit set of variables for each procedure call point within the reached procedure definition by

        i. defining a basic block live set for each block of computer code in a control flow graph for the reached procedure definition, the basic block live set comprising the variables used in the block of computer code and the variables used in any procedure called within the block of computer code, and

        ii. determining the live on exit set for each procedure call by taking the union of the basic block live sets for all successor blocks to the block in the control flow graph containing the procedure call point and by adjusting the

union to include uses of variables in the code between the call point for the procedure and the end of the block containing the call point.

b. storing the said live on exit set of variables for each procedure call point in an entry in a live on exit data structure comprising a bit vector indexed by a call graph edge,

c. defining a live on exit set of variables for the reached procedure definition by taking the union of all stored entries in the live on exit data structure corresponding to call points for the reached procedure,

d. removing all entries in the live on exit data structure corresponding to call points for the reached procedure, and

e. using the live on exit set of variables for the reached procedure definition to determine the variables that are ineligible for interprocedural dead store elimination in the reached procedure definition.

11. A computer program product for the compilation of computer code, the computer program product comprising a computer usable medium having computer readable code means embodied in said medium, comprising computer readable program code means to carry out the method steps of claim 1.

12. An optimizing compiler comprising

means for generating an intermediate representation of computer code, the intermediate representation comprising a call graph,

means for traversing the call graph in top down order,

means for storing a live on exit data structure,

means for generating a record in the live on exit data structure for each procedure call encountered in the traversal of the call graph, the record comprising data representing variables that are live at the point of the procedure call,

means for calculating the live on exit set for a procedure definition reached in traversing the call graph, the means for calculating the live on exit set comprising means for retrieving records from the live on exit data structure corresponding to the reached procedure definition and means for performing a union of the records to determine the live on exit set for the reached procedure definition, and means for signalling the availability of a dead store elimination optimization for a store operation contained in the reached procedure definition based on the live on exit set calculated for the procedure definition.

13. The optimizing compiler of claim 12, further comprising means for removing records associated with the reached procedure definition from the live on exit data structure following calculation of the live on exit set for the reached procedure definition.

14. A component for determining the correctness of a potential interprocedural dead store optimization for an optimizing compiler, the optimizing compiler generating an intermediate representation of code to be compiled comprising a call graph, the component comprising means to traverse the call graph in top-down order, and further comprising

means for defining a live on exit set of variables for each procedure call point within the reached procedure definition by

i. defining a basic block live set for each block of computer code in a control flow graph for the reached procedure definition, the basic block live set comprising the variables used in the block of computer code and the variables used in any procedure called within the block of computer code, and

ii. determining the live on exit set for each procedure call by taking the union of the basic block live sets for all successor blocks to the block in the control flow graph containing the procedure call point and by adjusting the union to include uses of variables in the code between the call point for the procedure and the end of the block containing the call point.

means for storing the said live on exit set of variables for each procedure call point in an entry in a live on exit data structure comprising a bit vector indexed by a call graph edge,

means for defining a live on exit set of variables for the reached procedure definition by taking the union of all stored entries in the live on exit data structure corresponding to call points for the reached procedure,

means for removing all entries in the live on exit data structure corresponding to call points for the reached procedure following definition of the live on exit set of variables for the reached procedure definition, and

means for determining the variables that are ineligible for interprocedural dead store elimination in the reached procedure definition, using the live on exit set of variables for the reached procedure definition.